Document Title	**Assignment 04**
Course	**CS 200 Introduction to Programming**
Academic Year	**2016-2017**
Semester	**Fall**
Due Date	**November 24, 2016 11:55 pm**
Marks	**100**

The assignment is due on **November 24, 2016 at 11:55 pm**. The late submission policy of 15% deduction per day up to 2 days applies. This assignment will require reasonable amount of time so try to start the assignment as early as you can.

Please keep in mind the following guidelines:

- Do not share your program code with anyone.
- Do not copy code from the internet.
- If you receive any assistance, mention the part of code in which you received assistance.
- You must be able to explain any part of your submitted code.
- All submissions are subject to automated plagiarism detection.

What to submit:

You have to make separate header files, source files and main file for each task. You have to submit all header files (.h files) and all .cpp files containing source code. Zip all .h files and .cpp files into one file named as <your 8 digit roll number>.zip and submit the zip file.

Task 1 Fractional Numbers (75 marks)

You have to implement a calculator which supports the calculation of fractional numbers. Each operation of the calculator consists of operand(s), result, number of operands, and the symbol of operation performed.

You have to implement a fraction class:

(40 Marks)

```
class fraction
{
private:
  int *numerator;
  int * denominator;
public:
  ……..
};
```

For fraction class, you need to implement the following:

- Default constructor
- Parameterized constructor
- Copy constructor
- Destructor
- Overload the following operators (+, –, *, /, >>, <<, ==, =, ++, – –)
  - + to add two fractional numbers
  - – to subtract two fractional numbers
  - * to multiply two fractional numbers
  - / to divide two fractional numbers
  - >> (input operator) to take input into fractional number from user
  - << (output operator) to output a fractional number
  - == (comparison operator) to compare two fractional number and will return true if both numbers are equal and false otherwise
  - = (assignment operator)
  - ++ (pre increment and post increment operator) it will return fractional number +1 as result e.g., incrementing 13/8 will result in 21/8
  - -- (pre decrement and post decrement operator) it will return fractional number -1 as result e.g ., decrementing 13/8 will result in 5/8

Then there is an operation class:

```
class operation                          (20 Marks)
{
  private:
     fraction *operands;
     fraction result;
     int operandCount;
     string symbol;
  public:
     ……….
};
```

For operation class, you have to write:

- Default constructor
- Parameterized constructor(s)
- Copy constructor
- Destructor
- Overload following operators (>>, <<)
    - << (output operator) to output operand(s), symbol of operation performed and result.
    - >> (input operator) to take input into operands of a particular operation

A user can:

- Add two fractional numbers using + operator
- Subtract two fractional numbers using – operator
- Multiply two fractional numbers using * operator
- Divide two fractional numbers using / operator
- Compare two fractional numbers using == operator
- Increment/decrement a fractional number using ++/-- operator

You have to use '<<' and '>>' operators to output and input fractional numbers respectively.

In the end you will have a class stack which contains list of performed operations in reverse chronological order.

```
class stack                                   (15 Marks)
{
    private:
       operation *list;
       int size;
    public:
           ……….
};
```

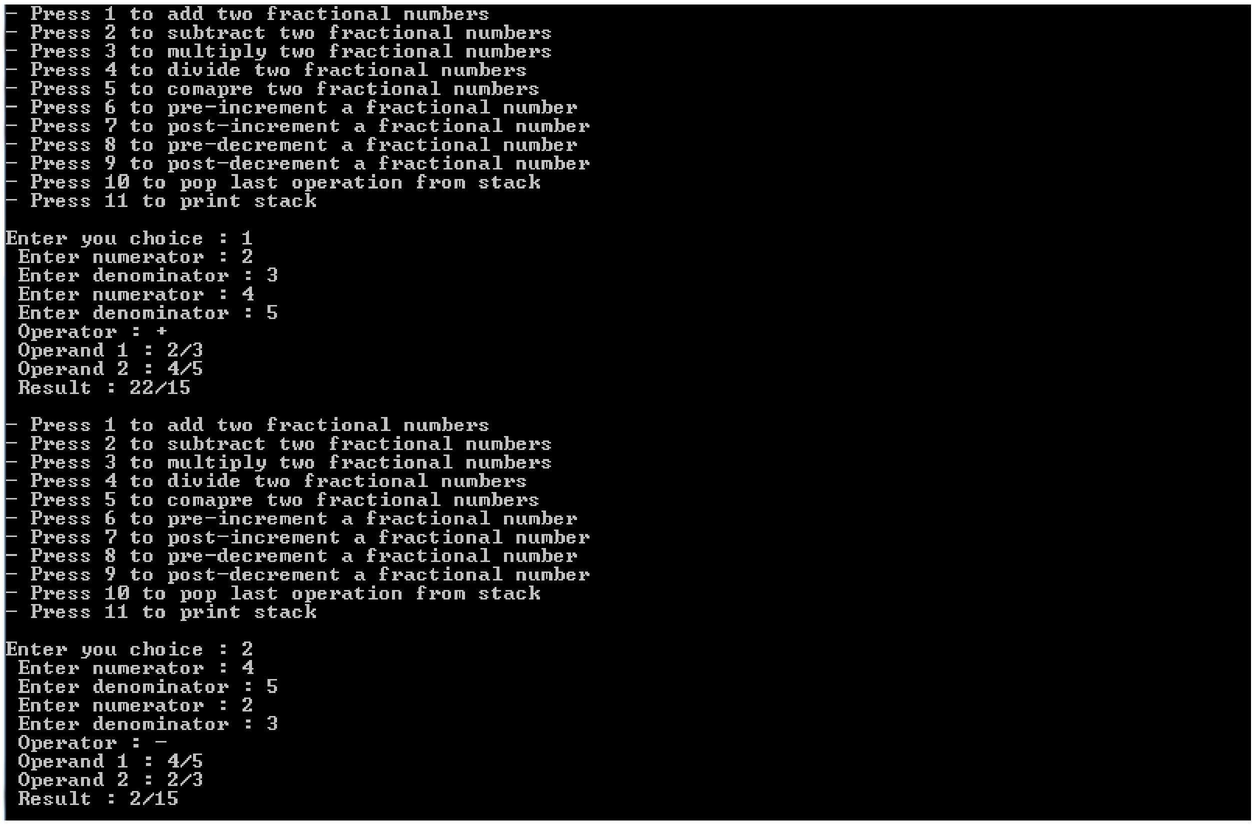
For stack class, you have to write:

- Default constructor
- Parameterized constructor (if necessary)
- Copy constructor (if necessary)
- Destructor
- Push function // used to insert an operation into list of operations
- Pop function // used to remove last inserted operation from list of operations
- Print function // used to print all list of operations in chronological order

Each performed operation will be pushed onto the stack (i.e., added to the list of operations) through the function push (….) and the function pop () that will remove the last inserted operation from the stack.

<u>Make sure that you are not doing 'shallow copy' at any point during this task and in stack class, the list of operations must be resized whenever a new operation is added or an operation is removed through the pop function.</u>

Sample Run:

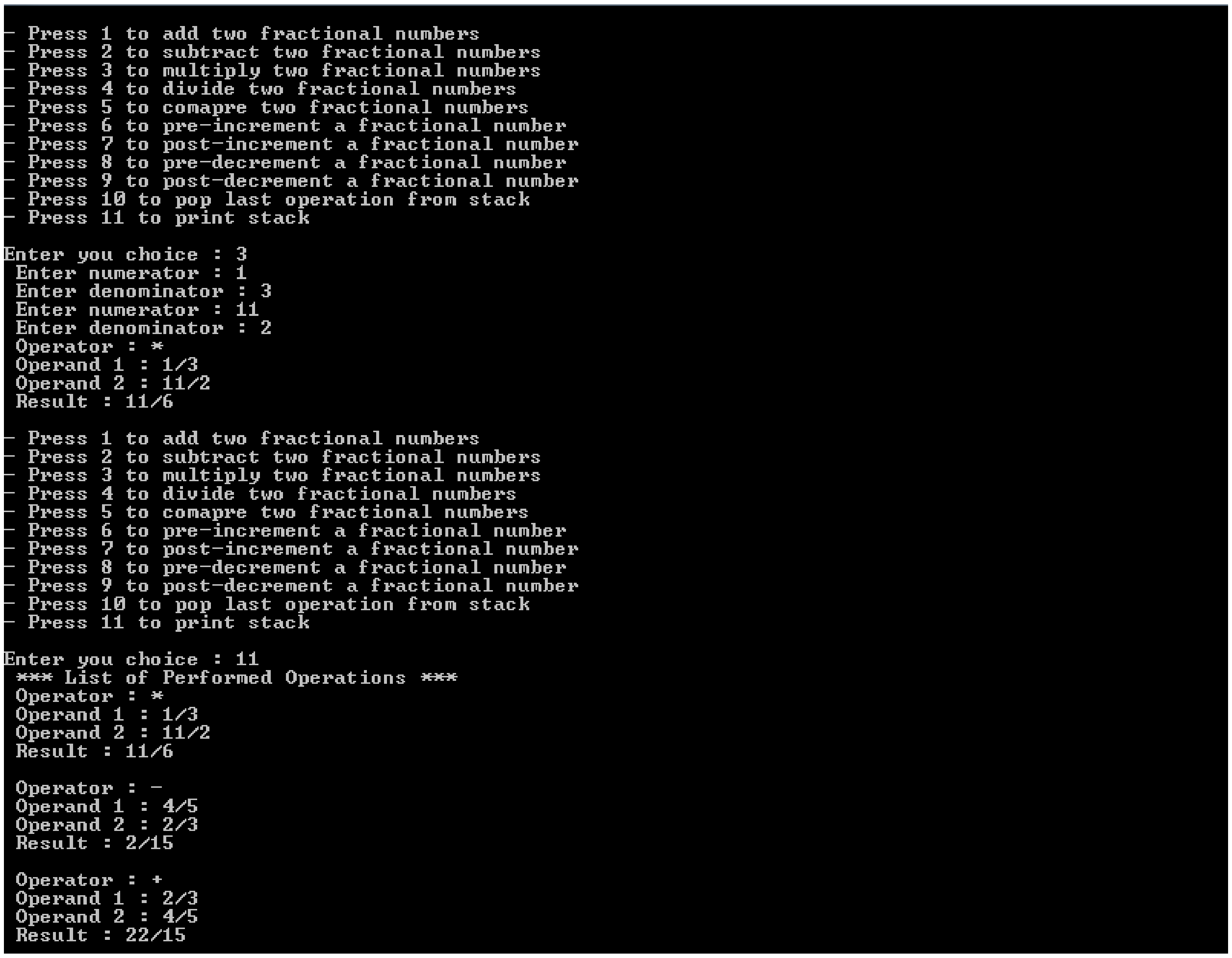


```
- Press 1 to add two fractional numbers
- Press 2 to subtract two fractional numbers
- Press 3 to multiply two fractional numbers
- Press 4 to divide two fractional numbers
- Press 5 to comapre two fractional numbers
- Press 6 to pre-increment a fractional number
- Press 7 to post-increment a fractional number
- Press 8 to pre-decrement a fractional number
- Press 9 to post-decrement a fractional number
- Press 10 to pop last operation from stack
- Press 11 to print stack

Enter you choice : 1
 Enter numerator : 2
 Enter denominator : 3
 Enter numerator : 4
 Enter denominator : 5
 Operator : +
 Operand 1 : 2/3
 Operand 2 : 4/5
 Result : 22/15

- Press 1 to add two fractional numbers
- Press 2 to subtract two fractional numbers
- Press 3 to multiply two fractional numbers
- Press 4 to divide two fractional numbers
- Press 5 to comapre two fractional numbers
- Press 6 to pre-increment a fractional number
- Press 7 to post-increment a fractional number
- Press 8 to pre-decrement a fractional number
- Press 9 to post-decrement a fractional number
- Press 10 to pop last operation from stack
- Press 11 to print stack

Enter you choice : 2
 Enter numerator : 4
 Enter denominator : 5
 Enter numerator : 2
 Enter denominator : 3
 Operator : -
 Operand 1 : 4/5
 Operand 2 : 2/3
 Result : 2/15
```



```
- Press 1 to add two fractional numbers
- Press 2 to subtract two fractional numbers
- Press 3 to multiply two fractional numbers
- Press 4 to divide two fractional numbers
- Press 5 to comapre two fractional numbers
- Press 6 to pre-increment a fractional number
- Press 7 to post-increment a fractional number
- Press 8 to pre-decrement a fractional number
- Press 9 to post-decrement a fractional number
- Press 10 to pop last operation from stack
- Press 11 to print stack

Enter you choice : 3
 Enter numerator : 1
 Enter denominator : 3
 Enter numerator : 11
 Enter denominator : 2
 Operator : *
 Operand 1 : 1/3
 Operand 2 : 11/2
 Result : 11/6

- Press 1 to add two fractional numbers
- Press 2 to subtract two fractional numbers
- Press 3 to multiply two fractional numbers
- Press 4 to divide two fractional numbers
- Press 5 to comapre two fractional numbers
- Press 6 to pre-increment a fractional number
- Press 7 to post-increment a fractional number
- Press 8 to pre-decrement a fractional number
- Press 9 to post-decrement a fractional number
- Press 10 to pop last operation from stack
- Press 11 to print stack

Enter you choice : 11
 *** List of Performed Operations ***
 Operator : *
 Operand 1 : 1/3
 Operand 2 : 11/2
 Result : 11/6

 Operator : -
 Operand 1 : 4/5
 Operand 2 : 2/3
 Result : 2/15

 Operator : +
 Operand 1 : 2/3
 Operand 2 : 4/5
 Result : 22/15
```

Task 2 Tic-Tac-Toe (25 marks)

Develop tic-tac-toe game in C++ using classes.

You have

- Square class which will represent box/position of board.
- Player class which will contain first name, last name and sign of player.
- Game class that will store a dynamic array of type square, two objects of player class and an integer to store dimension/size of board.

When the program starts, program will ask the user about the dimensions/size of board. Your program should support dimensions of (3, 5, 7, 9) only. Afterwards, the program will ask about the names of players and their respective symbols which will be used in game. Your program should randomly decide that which player will start the game. The player who succeeds in placing consecutive three of its marks horizontally, vertically or diagonally wins the game. If all the positions in grid are filled such that none of the players succeeds to place three consecutive marks horizontally, vertically or diagonally, game is said to be drawn.

You need to write the following for each class:

- Default constructor
- Parameterized constructor(s)
- Copy constructor (if required)
- Destructor (if required)

You have to overload ([][]) operator for game class. i-e if a player wants to play his move at 1st column and 1st row, then your program should support the following statement.

```
myGame[0][0]= myGame.getPlayerSymbol();

/* myGame is an object of game class and getPlayerSymbol() is a
function that returns symbol of a player */
```